Lecture #11

The Mobile-IoT Security Nexus

SMA 2025

The Mobile-IoT Security Nexus

When the Phone Becomes the Key to the Physical World

Today's Agenda

- **Part 1: The Control Plane** - Understanding the architecture of Mobile-IoT systems.
- **Part 2: REST Foundations** - Why the web's language is also the language of things.
- **Part 3: Authentication Protocols** - Moving beyond passwords to OAuth 2.0 and JWTs.
- **Part 4: Message Integrity** - HMACs, Signatures, and preventing Replay Attacks.
- **Part 5: The Threat Landscape** - How attackers pivot from the phone to the home.
- **Part 6: Case Studies** - Real-world examples of Mobile-IoT failures.

Recap from Lecture 10

- **Applied Crypto:** We learned about AES, RSA, and ECC.
- **Mobile Constraints:** We discussed battery, CPU, and the need for hardware acceleration.
- **TLS 1.3:** We saw how the handshake works and why it's faster.
- **Secure Storage:** We covered the Keystore and File-Based Encryption.

Today's Link: Lecture 10 gave us the *tools* (encryption, keys). Lecture 11 gives us the *system*. We will use those keys to authenticate API calls and sign commands sent to IoT devices.

Part 1: The Mobile Control Plane

The Phone as the Universal Remote

The Shift to "App-Centric" IoT

- **Old World:** IoT devices had physical buttons, screens, or dedicated remotes.
- **New World:** The device is "headless" (no screen). The User Interface (UI) is entirely on the smartphone app.
- **Implication:** If the app is insecure, the device is insecure. The app is the "Control Plane."

Architecture Model 1: Direct Connection

Phone <--> Device

- **Protocols:** Bluetooth Low Energy (BLE), Wi-Fi Direct, NFC.
- **Use Case:** Unlock a door, configure a device, stream audio.
- **Security Challenge:** Proximity-based attacks. The attacker must be physically near you.

Architecture Model 2: The Cloud Relay

Phone <--> Cloud <--> Device

- **Protocols:** HTTPS (Phone to Cloud), MQTT/WebSockets (Cloud to Device).
- **Use Case:** Turning on the heat while you are driving home. Viewing a security camera from work.
- **Security Challenge:** The "Attack Surface" is massive. It includes the App, the API, the Cloud Server, and the Device firmware.

Architecture Model 3: The Hybrid (Local + Cloud)

- **Concept:** The app tries to connect locally first (for speed). If that fails, it falls back to the cloud.
- **Complexity:** You need TWO authentication mechanisms.
  - One for the local network (e.g., a local token).
  - One for the cloud API (e.g., an OAuth token).

The "Trusted Controller" Concept

In all these models, the Mobile App is the **Trusted Controller**.

- The User authenticates to the App (Biometrics/PIN).
- The App authenticates to the Cloud/Device.
- **Critical Assumption:** The App is running on a secure, uncompromised phone.
- **The Threat:** If the phone is rooted or malware-infected, the "Trust" is broken.

Part 2: REST Foundations

The Language of the Internet of Things

What is REST?

- **REpresentational State Transfer.**
- **Architectural Style:** Not a strict protocol, but a set of constraints for building web services.
- **Key Principles:**
  - **Stateless:** Each request contains all the information needed to process it. The server doesn't remember the previous request.
  - **Resource-Based:** Everything is a "Resource" (e.g., a Lightbulb, a Thermostat, a User).
  - **Standard Methods:** Uses standard HTTP verbs (GET, POST, PUT, DELETE).

HTTP Verbs in IoT

How do we map IoT actions to HTTP?

- **GET:** Read the state.
- -> Returns *{ "power": "on", "brightness": 80 }*
- with body *{ "power": "off" }* -> Turns the light off.
- -> Creates a new automation routine.
- -> Revokes access.

JSON: The Data Format

- **JavaScript Object Notation (JSON).**
- **Why it rules IoT:**
  - Human-readable.
  - Lightweight (compared to XML).
  - Easy to parse on mobile (native support in Swift/Kotlin).

Statelessness and Scalability

- **The IoT Scale Problem:** A cloud server might manage millions of devices.
- **Stateful Server:** Keeping a connection open for every device is expensive (RAM/CPU).
- **Stateless REST:** The server handles a request and forgets it.
- **Benefit:** Infinite horizontal scaling. Just add more servers behind a Load Balancer.

Part 3: Authentication Protocols

Securing the API

The Evolution of API Auth

- **Basic Auth (The Bad Old Days):**
- Sending *username:password* in the header of every request.
  - **Risk:** If you can intercept one request, you have the user's password forever.
- A long random string sent in the header.
  - **Risk:** Hard to rotate. If stolen, the attacker has access until you manually regenerate it.
- OAuth 2.0 and JWTs.

OAuth 2.0: The Gold Standard

- **Concept:** Delegation of Authorization.
- **Roles:**
  - **Resource Owner:** You (the User).
  - **Client:** The Mobile App.
  - **Authorization Server:** The Cloud Identity Provider.
  - **Resource Server:** The IoT Cloud API.

The Access Token

- **What is it?** A digital key card.
- **Properties:**
  - **Short-lived:** Expires in 1 hour (usually).
  - **Scoped:** Valid only for specific actions.
  - **Revocable:** Can be cancelled by the server.

The Refresh Token

- **Problem:** If the Access Token expires every hour, does the user have to log in again?
- **Solution:** The Refresh Token.
  - A long-lived token used *only* to get new Access Tokens.
  - Stored securely in the Keystore/Keychain.

JWT (JSON Web Tokens)

- **Format:** *Header.Payload.Signature*
- **Stateless Auth:** The token *contains* the user's identity. The server doesn't need to look up the user in a database. It just verifies the signature.
- **Efficiency:** Perfect for high-scale IoT.

Anatomy of a JWT

- **Header:** Algorithm used (e.g., HS256 or RS256).
- **Payload (Claims):**
  - User ID (12345).
  - Expiration time.
  - "read:lights write:locks".

Validating a JWT

When the IoT Cloud receives a request:

- **Check Signature:** Does the signature match the content? (Proves integrity).
- **Check Expiration:** Is *now* < *exp*? (Proves validity).
- **Check Scope:** Does the token have the *write:locks* permission? (Proves authorization).

Common JWT Vulnerabilities

- **"None" Algorithm:** Attacker changes header to *{"alg": "none"}* and removes signature. Poorly written servers might accept it.
- **Secret Key Leak:** If the signing key is weak or leaked, attackers can forge their own tokens.
- **Lack of Expiration:** Tokens that last forever are a security nightmare.

Part 4: Message Integrity & Security

Protecting the Command

The Replay Attack

- **Scenario:**
  - User sends "Unlock Door" command via App.
  - Attacker records the encrypted network packet.
  - User leaves.
  - Attacker re-sends (replays) the exact same packet.
  - Door unlocks.

Defense 1: Timestamps

- **Mechanism:** Include the current time in the message.
- **Validation:** Server checks: *Is MessageTime within 5 seconds of ServerTime?*
- **Pros:** Simple.
- **Cons:** Requires clock synchronization. If the device clock drifts, valid commands are rejected.

Defense 2: Nonces

- **Nonce:** "Number used ONCE."
- **Mechanism:**
  - App generates a random number.
  - Includes it in the message.
  - Server remembers "I have seen Nonce X".
  - If Nonce X appears again, reject it.

HMAC (Hash-Based Message Authentication Code)

- **Goal:** Ensure the message hasn't been tampered with.
- **Mechanism:** *HMAC(Message, SecretKey)*.
- **Difference from Hash:** Uses a secret key. Only someone with the key can generate the valid HMAC.
- **Usage:** App sends *Message* + *HMAC*. Server calculates HMAC and compares.

Digital Signatures for High-Value Commands

- **Scenario:** "Unlock Smart Lock" or "Disable Alarm".
- **Requirement:** Non-repudiation. We need to prove *exactly* who sent the command.
- **Mechanism:**
  - App signs the command payload with the User's **Private Key** (stored in Secure Enclave).
  - Server verifies with User's **Public Key**.

Request Signing Implementation

```
// The Payload
{
  "command": "unlock",
  "device_id": "lock-123",
  "timestamp": 1678892000,
  "nonce": "a1b2c3d4"
}

// The Signature
Signature = Sign(SHA256(Payload), UserPrivateKey)

// The Header
Authorization: Bearer <AccessToken>
X-Signature: <Base64Signature>
```

Part 5: The Threat Landscape

Attacking the Home via the Phone

The "Pivot" Attack

- **Concept:** The phone is dual-homed. It is connected to the Internet (Cellular) and the Home Network (Wi-Fi).
- **The Attack:**
  - User installs a malicious "Flashlight App".
  - App requests "Local Network" permission.
  - App scans the Wi-Fi for vulnerable IoT devices (e.g., unpatched router, default password camera).
  - App sends commands to those devices.
  - App exfiltrates data back to the attacker via Cellular.

Malicious Apps & Permissions

- **Android/iOS Permission:** "Access Local Network".
- **Why it's dangerous:** It allows an app to talk to any device on your Wi-Fi.
- **User Behavior:** Users often grant this without thinking, especially for apps that claim to need it for "casting" or "setup".

DNS Rebinding

- **Goal:** Allow a malicious *website* to attack local IoT devices.
- **Mechanism:**
  - User visits *attacker.com*.
  - resolves to a real IP (1.2.3.4) initially.
  - Browser loads page and JavaScript.
  - DNS record changes (rebinding) to point *attacker.com* to *192.168.1.1* (Your Router).
  - JavaScript sends AJAX requests to *attacker.com*, which the browser now sends to your router.

Side-Channel Attacks via IoT

- **Scenario:** Smart Bulb.
- **Attack:**
  - Malware on phone monitors the *status* of the smart bulb.
  - If the bulb turns ON, the user is likely home/awake.
  - If the bulb turns OFF, the user is asleep/away.

The "Sleepy" Device Problem

- **Issue:** Battery-powered devices (sensors) sleep 99% of the time to save power.
- **Security Impact:** They cannot receive security updates immediately.
- **Attack:** Attacker exploits a vulnerability. Manufacturer releases a patch. Device is asleep. Attacker attacks before device wakes up to patch.

Part 6: Case Studies

Learning from Failure

Case Study 1: The Smart Aquarium

- **Target:** A Casino in Las Vegas.
- **Entry Point:** A smart thermometer in the lobby aquarium.
- **Vulnerability:** Default credentials, connected to the main corporate network.
- **The Pivot:** Attackers compromised the thermometer, pivoted to the network, and stole the High Roller database.
- **Lesson:** IoT devices must be isolated on a separate VLAN (Guest Network).

Case Study 2: Ring Camera Hacks

- **Incident:** Strangers talking to children through Ring cameras.
- **Cause:** Credential Stuffing (Users reusing passwords). Not a hack of Ring's servers.
- **Failure:** Lack of mandatory Multi-Factor Authentication (MFA).
- **Lesson:** MFA is mandatory for IoT security.

Case Study 3: Jeep Cherokee Hack (2015)

- **Target:** Connected Car.
- **Entry Point:** The Cellular connection to the Infotainment System (Uconnect).
- **The Pivot:** Attackers moved from the Infotainment system (Music/Maps) to the CAN Bus (Steering/Brakes).
- **Impact:** Remote control of the vehicle on a highway.
- **Lesson:** Network segmentation is life-critical.

Best Practices for Developers (1/3)

- **Enforce MFA:** Mandatory for all IoT accounts.
- **Use Standard Auth:** OAuth 2.0 / OIDC. Do not invent your own token system.
- **Validate Inputs:** Sanitize every JSON field sent to the device.

Best Practices for Developers (2/3)

- **Least Privilege:** The App should only ask for permissions it needs. The Token should only have scopes it needs.
- **Secure Storage:** Store Refresh Tokens in Keystore/Keychain.
- **Certificate Pinning:** Pin the connection between App and Cloud.

Best Practices for Developers (3/3)

- **Network Isolation:** Advise users to put IoT devices on a Guest Wi-Fi network.
- **Regular Audits:** Pentest the App AND the Device hardware.
- **Secure Defaults:** No default passwords. Random passwords printed on the sticker.

Future Trends: Matter Protocol

- **What is it?** A new unified standard for Smart Home (Apple, Google, Amazon).
- **Security:** Built-in.
- Mandatory encryption.
- Device attestation (Blockchain-style ledger of valid devices).
- Local control (works without cloud).

Future Trends: 5G and Edge Computing

- **5G:** Massive density of devices.
- **Edge:** Processing data closer to the device (not sending everything to the cloud).
- **Security:** Privacy improves (data stays local), but physical security of the Edge node becomes critical.

Appendix: OAuth 2.0 Flows

- **Authorization Code Flow:** Best for Mobile Apps.
- Uses a browser redirect.
- Securely exchanges code for token.
- Returns token in URL. Unsafe.

Appendix: MQTT QoS Levels

- **QoS 0:** At most once (Fire and forget).
- **QoS 1:** At least once (Guaranteed delivery, maybe duplicates).
- **QoS 2:** Exactly once (Slowest, most reliable).
- **Security:** QoS doesn't imply security, only reliability.

Appendix: BLE Pairing Modes

- **Just Works:** No user interaction. Unsecure against MitM.
- **Passkey Entry:** User types 6-digit code. Secure.
- **Numeric Comparison:** User compares numbers. Secure.
- **OOB (Out of Band):** NFC tap. Very secure.

Appendix: JSON Web Encryption (JWE)

- **JWT:** Signed (Integrity). Content is visible (Base64).
- **JWE:** Encrypted (Confidentiality). Content is hidden.
- **Use Case:** Passing sensitive data (PII) in a token.

Appendix: API Rate Limiting

- **Defense:** Prevent DoS attacks.
- **Mechanism:** "User X can only make 100 calls per minute."
- **Implementation:** API Gateway (Kong, AWS API Gateway).

Appendix: Zero Trust for IoT

- **Principle:** Never trust, always verify.
- **Network:** Micro-segmentation.
- **Identity:** Every device has a unique identity.

Appendix: Supply Chain Security

- **Risk:** Compromised libraries or hardware components.
- **SBOM:** Software Bill of Materials. Knowing what is in your code.

Appendix: Regulatory Landscape

- **IoT Cybersecurity Improvement Act (USA).**
- **ETSI EN 303 645 (Europe).**
- **Key Requirement:** No default passwords.

Appendix: Privacy by Design

- **Data Minimization:** Don't collect what you don't need.
- **Local Processing:** Process voice/video on the device, not the cloud.

Appendix: Responsible Disclosure

- **Bug Bounties:** Paying researchers to find bugs.
- **Policy:** How to report a vulnerability safely.

Appendix: Tools for IoT Hacking

- **KillerBee:** Zigbee analysis.
- **Ubertooth:** Bluetooth sniffing.
- **Binwalk:** Firmware analysis.

Part 7: Practical Code Examples

Talking to the Things

Android: Signed Request (OkHttp)

```
// Create the payload
val payload = JSONObject().apply {
    put("command", "unlock")
    put("timestamp", System.currentTimeMillis())
}

// Sign the payload (using our Crypto helper from Lecture 10)
val signature = signData(payload.toString().toByteArray())

// Build the request
val request = Request.Builder()
    .url("https://api.smart-home.com/devices/lock-123")
    .post(payload.toString().toRequestBody(JSON))
    .addHeader("Authorization", "Bearer $accessToken")
    .addHeader("X-Signature", signature) // Attach signature
    .build()

// Send it
client.newCall(request).execute()
```

iOS: Signed Request (URLSession)

```
var request = URLRequest(url: URL(string: "https://api.smart-home.com/devices/lock-123")!)
request.httpMethod = "POST"

// Create Payload
let payload = ["command": "unlock", "timestamp": Date().timeIntervalSince1970] as [String : Any]
let jsonData = try! JSONSerialization.data(withJSONObject: payload)
request.httpBody = jsonData

// Sign Payload (using CryptoKit)
let signature = try! privateKey.signature(for: jsonData)
let signatureString = signature.base64EncodedString()

// Attach Headers
request.setValue("Bearer \(accessToken)", forHTTPHeaderField: "Authorization")
request.setValue(signatureString, forHTTPHeaderField: "X-Signature")

URLSession.shared.dataTask(with: request).resume()
```

Android: MQTT Connect (Paho)

```
val clientId = MqttClient.generateClientId()
val client = MqttAndroidClient(context, "ssl://broker.hivemq.com:8883", clientId)

val options = MqttConnectOptions().apply {
    userName = "myUser"
    password = "myPassword".toCharArray()
    socketFactory = getSocketFactoryWithPinnedCert() // Important!
}

client.connect(options, null, object : IMqttActionListener {
    override fun onSuccess(asyncActionToken: IMqttToken?) {
        Log.d("IoT", "Connected to MQTT Broker")
        client.subscribe("home/livingroom/light", 1) // QoS 1
    }
    override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?) {
        Log.e("IoT", "Failed to connect")
    }
})
```

Summary of Lecture 11

- **The Nexus:** Mobile and IoT are inseparable. The phone is the controller.
- **REST is King:** Web standards (HTTP, JSON, OAuth) run the IoT world.
- **Auth Matters:** Tokens (JWTs) must be handled with extreme care.
- **Trust No One:** Verify signatures, check timestamps, and isolate networks.

Next Lecture Preview

Lecture 12: Mobile Application Pentesting

- We switch from "Builder" to "Breaker".
- Setting up a Pentest Lab (Genymotion, Corellium).
- Static Analysis (decompiling APKs).
- Dynamic Analysis (hooking with Frida).
- Bypassing SSL Pinning.

Q&A

Questions?